Slide 1

Administrivia

- Reminder: Homework 1 (short-answer questions about Dr. Lewis's guest lectures) due Monday.
- Homework 2 due in two parts: OpenMP program next Wednesday. MPI program the following Monday.

Slide 2

MPI — the Message Passing Interface

- Idea was to come up with a single standard (concepts and library) for message-passing programs, then allow many implementations. Similar to language standards (C, C++, etc.). Good for portability.
- MPI Forum — international consortium — began work in 1992. First standard MPI 1.1, followed by MPI 2.x and 3.x. 1.1 specification is 500+ pages, and later standards even bigger.
- Original reference implementation — MPICH (Argonne National Lab). LAM/MPI (Local Area Multicomputer) is another free implementation. Latest / most popular may be OpenMPI (installed on department machines).

  (Yes — OpenMP, OpenMPI, very confusing! but aside from names, unrelated.)

Slide 3

What's an MPI Program Like?

- "SPMD" (Single Program, Multiple Data) model — many processes, all running the same source code, but each with its own memory space and each with a different ID. Could take different paths through the code depending on ID.
- Source code in C/C++/Fortran, with calls to MPI library functions.
- How programs get started isn't specified by the (first) standard! (for historical/political reasons — some early target platforms were very restrictive, would not have supported what academic-CS types wanted).
- (Compare and contrast all of the above with OpenMP.)

Slide 4

What's in the MPI Library?

- Setup and bookkeeping — initialization, cleanup, environment query, etc.
- Data management — pack/unpack, derived data types.
- Point-to-point communication — several varieties, differing mostly in how much synchronization.
- Collective operations — e.g., broadcast.
- More . . .

Slide 5

MPI "Communicators"

- (One more thing to define before we can write simple code.)
- MPI allows grouping processes; group plus associated context called a "communicator". Makes it easier to write "safe" parallel libraries.
- Predefined communicator `MPI_COMM_WORLD` includes all processes. Programmers can create additional ones.

Slide 6

Compiling and Running MPI Programs — Setup

- OpenMPI starts processes on remote machines using SSH. In order for this to work, your account has to be set up to not prompt for a password.
- You can find instructions for setting that up linked from my home page `http:/www.cs.trinity.edu/~bmassing`.

Slide 7

Compiling and Running MPI Programs — Setup Continued

- Also be aware that OpenMPI commands and functions not part of default search path. To use them you need either

  `module load openmpi-default`

  (for the Scientific Linux default version) or

  `module load openmpi-latest`

  (for a locally-compiled version using the latest GCC and enabling more features).
- Note that once you load one of these modules, you should have access to `man` pages for all MPI commands *and functions*.

Slide 8

Compiling MPI Programs

- Compile with `mpicc`. (I say use my `make` file.)
- (`mpicc` basically invokes `gcc` with some extra parameters to access the MPI include files and libraries.)

Slide 9

Running MPI Programs

- *Can* just call executable, but that only launches one process.
- Instead, use `mpirun`. Many many options, so very flexible, but also can be difficult to figure out how to get it do what you want.
- Very basic usage (to start two processes):

```
mpirun -np 2 ./hello
```

- But this starts all processes on the same machine . . .

Slide 10

Running MPI Programs, Continued

- Various ways to specify where to start processes:
  - `-host` followed by comma-separated list of values
    Note however that you may need to have one machine name per process.
  - `-hostfile` followed by name of a file containing machine names.
    Note however that by default this tends to bunch up processes on first few machines listed. To spread them around more, add `-map-by node`. (This is my preference, but might be worth trying both ways and comparing performance!)
- Also, remotely-launched programs may have trouble finding MPI library code. A way to resolve that is with `-prefix`.
- My script `run-pgm` may be useful.

Slide 11

Simple Example(s)

- Look at "hello world" proggram; compile and run.

Slide 12

Simple (Blocking) Point-to-Point Communication in MPI

- Send with `MPI_Send` — returns as soon as data has been copied to system buffer, buffer in program can be reused.
- Receive with `MPI_Recv` — waits until message has been received.
- Can use "tags" to distinguish between kinds of messages. Can receive selectively or not (`MPI_ANY_TAG`). Received tag is in returned `MPI_Status` variable (e.g., `status.MPI_TAG`).
- Can receive from specific sender or from any sender. (`MPI_ANY_SOURCE`). Sender is in returned `MPI_Status` variable (e.g., `status.MPI_SOURCE`).
- For `MPI_Recv`, "length" parameter specifies buffer length. Use `MPI_Get_count` to get actual count.
- Look at sample program `send-recv.c`.

Slide 13

Not-So-Simple Point-to-Point Communication in MPI

- For not-too-long messages and when readability is more important than performance, `MPI_Send` and `MPI_Recv` are probably fine.
- If messages are long, however, buffering can be a problem, and can even lead to deadlock. Also, sometimes it's nice to be able to overlap computation and communication.
- Therefore, MPI offers several other kinds of send/receive functions: "synchronous" (blocks both sender and receiver until communication can take place), "non-blocking" (doesn't block at all, program must later test/wait for communication to take place).

  (More about these later.)

Slide 14

Collective Communication in MPI

- "Collective communication" operation — one that involves many processes (typically all, or all in MPI "communicator").
- Could implement using point-to-point message passing, but some operations are common enough to be library functions — broadcast (`MPI_Bcast`), "reduction" (`MPI_Reduce`), etc.

Slide 15

Numerical Integration, Revisited

- Recall numerical integration example, sequential version.
- How to parallelize with MPI? Can we use sort of the same strategy we used for OpenMP, or do we need something totally different?

  (Pause the video and think about it a few minutes. Record your ideas.)

Slide 16

Numerical Integration in MPI

- Same basic strategy we used for OpenMP — split up loop iterations among UEs, have each compute local sum, combine at the end — will work here too. But there are some differences:
- OpenMP has nice syntax for splitting loop iterations among threads; programmer doesn't need to do this explicitly. Not so with MPI.
- With no shared memory, no worries about shared variables. But combining partial results is more work.
- Sample program `num-int-par.c`.

Slide 17

A Few Words About Measuring Performance

- For most if not all programs we write for this class, we'll be interested in finding out how they "scale" with varying numbers of UEs. To make this interesting you need to try it on a platform where you can vary that a lot. Classroom machines are probably not ideal for this; `Dione` (old and slow but *lots* of cores) and the `Pandora` cluster better. For specifics of all classroom/lab machines, see "Specifics" in

  `https://sites.google.com/trinity.edu/csci-department-computers/`

- Probably smart to re-run experiments at least twice so you have some idea of whether times are repeatable. To be really careful should probably run several times (four? five?) and average.

Slide 18

Minute Essay

- What were your initial thoughts about how to "parallelize" the numerical integration example in MPI? does what I propose make sense to you?
- Questions?